Современные системы поддержки доказательств и их применения
(ФИЦ РАН, 28 февраля 2025 г.)

Sergei Soloviev, IRIT, Toulouse
roschinka1956@gmail.com

28 февраля 2025

Системы автоматического доказательства

- Первой работающей системой была, насколько мне известно, система, разработанная в ЛОМИ АН СССР: Шанин Н.А., Давыдов Г.В., Маслов С.Ю., Минц Г.Е., Оревков В.П., Слисенко А.О. Алгорифм машинного поиска естественного логического вывода в исчислении высказываний/АН СССР. Матем. ин-т им. В.А. Стеклова. Ленингр. отд-ние-Москва-Ленинград:Наука. Ленингр. отд-ние, 1965. - 39 с.
- Чуть позже появилась система Automath, разработанная голландским логиком De Bruijn'ом: N.G. de Bruijn. Automath : A Language for Mathematics. EUT Report. WSK, Dept. of Mathematics and Computing Science. Technische Hogeschool Eindhoven, 1968.
- Нельзя не упомянуть также язык PROLOG (1972, A. Colmerauer et Ph. Russell).

Системы автоматического доказательства

- Системы автоматического доказательства - ATP (Automatic Theorem Provers) - получают на входе некоторое утверждение, обычно выраженное на формальном логико-математическом языке и действуют как “черный ящик” (т.е. без вмешательства человека на промежуточных этапах) с целью либо получить доказательство истинности утверждения, либо построить контрпример.
- ATP, разработанный в ЛОМИ группой Н.А. Шанина в 1965 г., действовал в рамках исчисления высказываний.
- ATP часто разрабатывались для формул языка логики первого порядка, т.е. включающего кванторы всеобщности $\forall$ и существования $\exists$, переменные, предикатные символы, обычно также $=$. (Например Vampire, Superposition.)

Системы автоматического доказательства

- Интересная деталь - программа АЛПЕВ, разработанная группой Шанина , уже тогда реально работала на вычислительной машине “Урал”.
- То, что система первоначально разрабатывалась для исчисления высказываний, разумеется, не означает, что не велось работы по её расширению на исчисление предикатов.
- Но прежде чем говорить о других видах систем поддержки доказательства, есть смысл остановиться на основных приниципах и трудностях, связанных с автоматическим (и интерактивным) поиском док-в и их проверкой.

Принципы и трудности

Приведем правила естественного вывода (natural deduction) для исчисления высказываний. Как обычно, Γ слева означает список формул (гипотез), E - правила удаления, I - правила введения связок.

$$\frac{A \in \Gamma}{\Gamma \vdash A}(\text{axiom}) \quad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}(\text{E}\to) \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}(\text{I}\to)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}(\text{E}\wedge_1) \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}(\text{E}\wedge_2) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}(\text{I}\wedge)$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}(\text{E}\vee) \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}(\text{I}\vee_1) \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}(\text{I}\vee_2)$$

Надо сказать, что правила для отрицания выглядят несколько менее естественными.

Принципы и трудности

- $$\frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \bot}(E\neg)$$

  (это правило можно было бы также рассматривать как введение противоречия $\bot$),

- $$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}(I\neg) \quad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}(E\,\bot)$$

  (справа - из лжи следует все, что угодно).

- Особую роль играет закон исключенного третьего $\Gamma \vdash A \vee \neg A$, его включают в качестве аксиомы в случае т.н. классической логики и исключают, когда рассматривается интуиционистская.
- Можно отметить, что в этих правилах могут появляться и исчезать формулы - например, A.

Принципы и трудности

- Пример вывода (и поиска вывода).
- Как вывести формулу $\vdash \neg(p \to q) \to (\neg p \to r)$?
- С точки зрения семантики (таблиц истинности) все тут довольно понятно: если импликация $p \to q$ неверна, p должно быть истинно, а тогда из отрицания p следует все что угодно.
- Поиск идет с конца к началу:

$$\dfrac{\dfrac{\dfrac{\neg(p \to q), \neg p \vdash \bot}{\neg(p \to q), \neg p \vdash r}}{\neg(p \to q) \vdash \neg p \to r}}{\vdash \neg(p \to q) \to (\neg p \to r)}.$$

- Струкутура формулы подсказывает, что два правила в конце вывода - это I $\to$, а перед этим - откуда взяться единственному вхождению r? Подходящим правилом выглядит E $\bot$.

Принципы и трудности

- Но откуда тогда берется $\bot$? Подходящим правилом выглядит "закон противоречия" E$\neg$. Правда, не все так просто.
-

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(p \to q), \neg p \vdash \neg(p \to q)}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{}{\neg(p \to q), \neg p, p \vdash \neg p}\,\text{ax} \qquad \dfrac{}{\neg(p \to q), \neg p, p \vdash p}\,\text{ax}}{\neg(p \to q), \neg p, p \vdash \bot}}{\neg(p \to q), \neg p, p \vdash q}}{\neg(p \to q), \neg p \vdash p \to q}}{\neg(p \to q), \neg p \vdash \bot}$$

- Правило E$\neg$ пришлось использовать дважды!

- В отличие от семантической верификации (теории моделей), в системах поиска вывода поиск опирается на синтаксический анализ формул - по структуре формулы ищется, какое правило могло быть применено, чтобы дать искомый результат.
- И в этом примере мы видим по крайней мере два трудных места для этого подхода, где не вполне ясно, из каких именно формул могло появиться противоречие $\perp$.
- В связи с трудностями этого рода с самого начала было понятно, что выбор системы правил для поиска вывода играет очень важную роль.
- Не удивительно, поэтому, что с самого начала рассматривались дополнительные правила, которые не меняют класса выводимых формул, но могут ускорить поиск вывода.

Принципы и трудности

- Примером таких правил служат так называемые производные и допустимые правила.
- Производное правило - это комбинация (фиксированная) основных правил, позволяющая “укоротить” вывод
- Допустимое правило не выражается, вообще говоря, таким образом, но не добавляет новых выводимых формул.
- Примеры: $\dfrac{\Gamma \vdash B}{\Gamma, A \vdash B}$ (“утончение”), $\dfrac{\Gamma \vdash B \quad \Gamma, B \vdash C}{\Gamma \vdash C}$ (“сечение”).
- По отношению к системе естественного вывода первое правило допустимое, но не производное, а второе - производное.
- В случае утончения надо добавить A ко всем аксиомам (и далее A сохранится до конца вывода).
- Сечение представляется в виде

$$\dfrac{\dfrac{\Gamma, B \vdash C}{\Gamma \vdash B \to C}(I \to) \quad \Gamma \vdash B}{\Gamma \vdash C}(E \to).$$

Принципы и трудности

- Применение допустимых правил позволяет еще более укоротить вывод, но может быть проблемным, т.к. какая-то важная фомула может отсутствовать в заключении правила, и как тогда ее искать в процессе поиска вывода “снизу вверх”, от заключений к посылкам правил?
- Рассмотрим, для сравнения, другую, альтернативную систему правил, правила так называемого секвенциального исчисления, предложенного Г. Генценом (1934).
- Заметим, что и эта система, и система “естественного вывода” (и ряд других - например “гильбертовские” системы) были предложены еще до постановки задачи автоматического поиска вывода.

Принципы и трудности

В отличие от системы естественного вывода, здесь правила делятся на правила введения связок слева и справа от $\vdash$. Единственным различием между классической и интуиционистской логикой является условие, что в интуиционистском случае справа от $\vdash$ должно быть не более одной формулы. (Более “структурный” взгляд.)

$$\frac{A \in \Gamma}{\Gamma \vdash A}(\text{axiom}) \quad \frac{\Gamma \vdash \Theta, A \quad B, \Delta \vdash \Lambda}{\Gamma, A \to B, \Delta \vdash \Theta, \Lambda}(\to\vdash) \quad \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}(\vdash\to)$$

$$\frac{A, \Gamma \vdash \Theta}{A \wedge B, \Gamma \vdash \Theta}(\wedge_1\vdash) \quad \frac{B, \Gamma \vdash \Theta}{A \wedge B, \Gamma \vdash \Theta}(\wedge_2\vdash) \quad \frac{\Gamma \vdash \Theta, A \quad \Gamma \vdash \Theta, B}{\Gamma \vdash A \wedge B}(\vdash\wedge)$$

$$\frac{\Gamma, A \vdash \Theta \quad \Gamma, B \vdash \Theta}{\Gamma, A \vee B \vdash \Theta}(\vee\vdash) \quad \frac{\Gamma \vdash \Theta, A}{\Gamma \vdash \Theta, A \vee B}(\vdash\vee_1) \quad \frac{\Gamma \vdash \Theta, B}{\Gamma \vdash \Theta, A \vee B}(\mathrm{I}\vee_2)$$

$$\frac{A, \Gamma \vdash \Theta}{\Gamma \vdash \Theta, \neg A}(\vdash\neg) \quad \frac{\Gamma \vdash \Theta, A}{\neg A, \Gamma \vdash \Theta}(\neg\vdash)$$

Принципы и трудности

- Понятно, что при таком более структурном подходе исчисление содержит и больше структурных правил (не вводящих и не удаляющих связок).
- Утончение: $\dfrac{\Gamma \vdash \Theta}{D, \Gamma \vdash \Theta}$, $\dfrac{\Gamma \vdash \Theta}{\Gamma \vdash \Theta, D}$.
- Сокращение: $\dfrac{D, D, \Gamma \vdash \Theta}{D, D, \Gamma \vdash \Theta}$, $\dfrac{\Gamma \vdash \Theta, D, D}{\Gamma \vdash \Theta, D}$.
- Сечение: $\dfrac{\Gamma \vdash \Theta, D \quad D, \Delta \vdash \Lambda}{\Gamma, \Delta \vdash \Theta, \Lambda}$.
- Замечание. Детали, кажущиеся второстепенными с точки зрения человека, могут оказаться очень важными при автоматическом поиске вывода. Например, являются ли $\Gamma, \Delta, \Theta, \Lambda$ множествами или списками формул, сильно влияет на сложность поиска.

Принципы и трудности

- Вывод $\vdash \neg(p \to q) \to (\neg p \to r)$ в новом исчислении:
- $$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash p}(\neg \vdash)}{\neg p, p \vdash}(\vdash \text{wkn})}{\neg p, p \vdash q}(\vdash \to)}{\neg p \vdash p \to q}(\neg \vdash)}{\neg(p \to q), \neg p \vdash}(\vdash \text{wkn})}{\neg(p \to q), \neg p \vdash r}(\vdash \to)}{\neg(p \to q) \vdash \neg p \to r}(\vdash \to)}{\vdash \neg(p \to q) \to (\neg p \to r)}(\vdash \to).$$
- Выводы в секвенциальном исчислении Генцена (без сечения) обладают свойством подформульности - всякая формула, появляющаяся в выводе где-либо, входит также в конечную секвенцию. Это очень удобно при поиске вывода.
- (А правило сечения относится к числу допустимых.)

Принципы и трудности

- Уже этих примеров достаточно, чтобы очертить основные трудности, с которыми встречается автоматический поиск логического вывода.
- Основной принцип ATP - это поиск вывода на основе структуры формул.
- Программа ищет, исходя из структуры формулы, результатом применения каких правил вывода может быть данная формула. Поиск идет от заключений правил к посылкам и в случае успеха завершается аксиомами.
- В исчислении высказываний принципиальные трудности, с которым он сталкивается, носят сложностной характер. Даже при удачном выборе логической системы (например, со свойством подформульности) приходится ожидать экспоненциальной сложности (по отношению к размеру формул).

Принципы и трудности

- Вспомним, что таблица истинности для формулы исчисления высказываний от n переменных содержит 2^n строк.
- Выигрыш, при поиске вывода, может быть связан с особенностями структуры, которые не зависят от количества переменных. Но в целом порядок сложности тот же.
- Все же мы остаемся в рамках конечного.
- При переходе к исчислению предикатов в игру входит бесконечность - например, значения переменных могут принадлежать бесконечным множествам. И это, разумеется, накладывает свой отпечаток на проблему поиска вывода.

Принципы и трудности

- Известный пример, иллюстрирующий сразу многие виды трудностей.
- Теорема. Существуют иррациональные числа x, y, такие, что x^y является рациональным числом.
- Доказательство. Известно, что $\sqrt{2}$ - иррациональное число. Число $\sqrt{2}^{\sqrt{2}}$ либо является, либо не является рациональным (TND). Если это число рациональное, теорема доказана. Можно взять $x = \sqrt{2}, y = \sqrt{2}$. Если это число иррациональное, возьмем $x = \sqrt{2}^{\sqrt{2}}$ и $= \sqrt{2}$. Тогда

$$x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2.$$

Принципы и трудности

- Трудности.
- Заметим, что это доказательство предполагает уже известным, что $\sqrt{2}$ - иррациональное число.
- Если вам скажут, что модный сейчас AI сам легко найдет его - не верьте, оно (или более общее, или близкое - аналогичное) содержится в базе данных, к которой он подключен.
- Впрочем, довольно естественно, что база данных по математике содержит много известных на сегодняшний день иррациональных чисел.
- Замечание. Интересно (проведите эксперимент на себе), много ли иррациональных чисел мы знаем по именам?

Принципы и трудности

- Взяв какое-нибудь из этих иррациональных чисел, x можно рассмотреть x^x.
- Но идея доказательства состоит в том, что если x^x тоже иррационально, из этого числа должно быть легко (комбинируя его с x или другим известным иррациональным числом) получить рациональное число.
- Т.е. опять в игру вступает комбинаторика, что всегда трудно для автоматических систем.
- Наконец, полученное доказательство неконструктивно - использование TND не дает способа определить, какая именно пара иррациональных чисел является подходящей.

Принципы и трудности

- При переходе к исчислению предикатов (и к логикам высших порядков) появляются и другие источники неконструктивности.
- $\dfrac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[s/x]}(E\forall)$
  Здесь s - произвольный терм, т.е. произвольное “объектное выражение” из, вообще говоря, бесконечного множества.
- $\dfrac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}(I\forall)$
  (в $I\forall$ $x \notin FV(\Gamma)$).
- $\dfrac{\Gamma \vdash A[s/x]}{\Gamma \vdash \exists x.A}(I\exists)$.
- $\dfrac{\Gamma \vdash \exists x.A \quad \Gamma, A \vdash C}{\Gamma \vdash C}(E\exists)$
  Здесь формула A исчезает (не входит в заключение правила), что может оказаться существенной помехой при поиске вывода.

Интерактивные системы поддержки доказательства

- Интерактивные системы поддержки доказательства (ITP, Interactive Theorem Provers), в отличие от ATP, предназначены для проверки и дополнения доказательств, написанных человеком на специально созданных для этой цели языках программирования
- Эти языки позволяют запись и последующую обработку (processing) формальных доказательств.
- С логической точки зрения, от "встроенных" языков требуется высокий уровень выразительности, по отношению к этим языкам обычное исчисление высказываний или исчисление предикатов представляют частные случаи.
- Обычно это языки логик высших порядков (HOL, Higher Order Logic) или теорий зависимых типов (DETT, Dependent Type Theories).

ITP

- В ITP система показывает текущее состояние доказательства (целевую формулу и гипотезы).
- Пользователь может указывать направление поиска (например, выбрать одну из гипотез и развивать ее доказательство, выбрать одно из возможных правил).
- Система может по его указанию развернуть несколько шагов доказательства, составить список случаев и т.п.
- Начиная с 1960-х, был разработан ряд ITP (их также часто называют Proof Assistants). Эти системы различаются размерами “ядра”, логическими системами, положенными в основу, и степенью выразительности соотвествующих языков.
- Из наиболее известных:

ITP

- Mizar (R. Matuszewski P. Rudnicki. MIZAR: The first 30 years. Mechanized Mathematics and Its Applications, 4, 3-24, 2005.)
- Agda (L. Magnusson and B. Nordstrom. The Alf proof editor and its proof engine. In G. Goos, J. Hartmanis, H. Barendregt, and T. Nipkow, eds, Types for Proofs and Programs, v. 806, 213-237. Springer, 1994).
- Coq (T. Coquand PhD Thesis (1985), P. Castéran and Y. Bertot. Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. 2004.)
- HOL-Light (M. Gordon. From LCF to HOL: A Short History. In G. Plotkin, C. P. Stirling, and M. Tofte, eds, Proof, Language, and Interaction, 169-186. The MIT Press, May 2000.)
- Isabelle (T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL | A Proof Assistant for Higher-Order Logic, LNCS, 2883. 2002.)
- Lean (L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean Theorem Prover (System Description). CADE-25, 378-388, Cham, Springer 2015.)

ITP

- Рассмотрим несколько примеров с сайта, посвященного языку Coq, https://coq.vercel.app/
- Страница посвящена jsCoq, интерактивному сетевому варианту Coq (open source).
- (Для экспериментов с jsCoq на сегодняшний день больше подходят некоторые другие страницы, например https://jscoq.github.io/wa/.)

Пример

- Разворот (обращение) списка: $\mathrm{rev} \circ \mathrm{rev} = \mathrm{id}$.
- Подготовка:

```
1 From Coq Require Import List.
2 Import ListNotations.
```

- Доказательство использует лемму:

```
3 Lemma rev_snoc_cons A:
4 ∀(x:A)(l:listA), rev(l ++ [x]) = x::rev l.
```

cons - добавление элемента, snoc - добавление элемента с другого конца.

- Лемма доказывается стандартной индукцией:

```
5 Proof.
6   induction l.
7       – reflexivity.
8       – simpl. rewrite IHl. simpl. reflexivity.
9 Qed.
```

Пример

- Основное утверждение:
  Theorem rev_rev A:∀(l:listA), rev(rev l) = l.
  Proof.
  induction l.
  −reflexivity.
  −simpl. rewriterev_snoc_cons. rewrite IHl.
  reflexivity.
  Qed.
- Другие примеры включают, скажем, иррациональность $\sqrt{2}$.

Индуктивные типы

- Важной чертой современных ITP является использование индуктивных типов.
- Элементы индуктивных типов строятся рекурсивно, с помощью “конструкторов”. Задание индуктивного типа в Coq и других ITP включает соответствующий оператор рекурсии (для обределения функций) и аксиому индукции (индукция “по построению” элементов).
- Мы только что видели тип list, другие индуктивные типы - это тип натуральных чисел N, типы деревьев (бинарных или, скажем, ω-деревьев).

Индуктивные типы

- Рассмотрим несколько примеров (на “полуформальном” языке).
- Тип натуральных чисел $\mathrm{N} = \mathcal{M}(\mathrm{X}, (\mathrm{X})\mathrm{X})$. Здесь X, (X)X - индуктивные схемы, определяющие способ порождения элементов. Схема X задает просто константу, элемент индуктивного типа (в случае N подразумеватеся 0), (X)X задает операцию порождения новых элементов, в данном случае successor $\mathrm{s}{:}\mathrm{N} \to \mathrm{N}$.
- Тип списков над A, $\mathrm{List}{:}[\mathrm{A}{:}\mathrm{Type}]\mathcal{M}(\mathrm{X}, (\mathrm{A})(\mathrm{X})\mathrm{X})$. Схема X вновь задает константу, на это раз подразумевается пустой список. Схема (A)(X)X соответствует операции от двух аргументов, $\mathrm{cons}(\mathrm{a}, \mathrm{x})$.
- Тип бинарных деревьев (с элементами типа A в вершинах) задается как $\mathrm{BT} = [\mathrm{A}{:}\mathrm{Type}]\mathcal{M}(\mathrm{X}, (\mathrm{A})(\mathrm{X})(\mathrm{X})\mathrm{X})$. Здесь константа - пустое дерево , а операция задает по двум деревьям и элементу a новое дерево $\mathrm{join}(\mathrm{a}, \mathrm{T}_1, \mathrm{T}_2)$.

Индуктивные типы

- Аргументом при построении элементов некоторых типов могут служить и функции, например, в случае ω-деревьев.
- $\mathrm{T}\omega = [\mathrm{A{:}Type}]\mathcal{M}(\mathrm{X}, (\mathrm{A})((\mathrm{N})\mathrm{X})\mathrm{X})$. Здесь по функции $\mathrm{f{:}N} \to \mathrm{T}\omega$ и $\mathrm{a} \in \mathrm{A}$ строится новое дерево с вершиной a с ветвями, занумерованными $\mathrm{n} \in \mathrm{N}$ (n-е поддерево $\mathrm{f(n)}$).
- "Конструкторы" элементов рассматриваются как операторы введения, ассоциированные с индуктивным типом. Есть и оператор удаления (снятия "конструктора"), соответствующий шагу индуктивного доказательства по построению и рекурсивного определения функций на индуктивном типе.
- Например для N это $\mathrm{Rec_N{:}(C{:}(N)Type)(c{:}C(0))(f{:}(x{:}N)(C(x))C(s(x)))(z{:}N)C(z)}$.
- Вместе с ним вводятся два правила для равенства:

$$\mathrm{Rec_N}(\mathrm{C}, \mathrm{c}, \mathrm{f}, 0) = \mathrm{c{:}C}(0)$$

$$\mathrm{Rec_N}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \mathrm{s}(\mathrm{n})) = \mathrm{f}(\mathrm{n}, \mathrm{Rec_N}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \mathrm{n}))\mathrm{:C}(\mathrm{s}(\mathrm{n})).$$

Типа результата в ходе рекурсии может меняться.

Индуктивные типы

- В случае бинарных деревьев
- $\mathrm{Rec_{BT}}$:(C:(BT)Type)(c:C(λ))
  (f:(x:BT)(y:BT)(a:A)(C(x))(C(y))(C(join(x, y, a))))(z:BT)C(z)
- $\mathrm{Rec_{BT}}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \lambda) = \mathrm{c}{:}\mathrm{C}(0)$
- $\mathrm{Rec_{BT}}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \mathrm{join}(\mathrm{x}, \mathrm{y}, \mathrm{a})) =$
  $\mathrm{f}(\mathrm{x}, \mathrm{y}, \mathrm{a}, \mathrm{Rec_{BT}}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \mathrm{x}), \mathrm{Rec_{BT}}(\mathrm{C}, \mathrm{c}, \mathrm{f}, \mathrm{y}))$

Индуктивные типы

- Все эти операторы и правила для равенства задаются автоматически по заданию индуктивного типа.
-
-
-

Взаимодествие с генеративным ИИ

- Достаточно интересный (открытый) вопрос - перспективы взыимодействия с генеративным ИИ.
- В принципе, системы генеративного ИИ могут использоваться для генерации текстов.
- В том числе - программ.
- Или - доказательств (понимаемых как частный случай текстов).

Взаимодествие с генеративным ИИ

- Однако в нескольких широко разрекламированных случаях трудно отделить использование огромных математических баз данных от собственно “генеративной” составляющей.
- В нескольких случаях, которые мне известны не из прессы, а непосредственно от коллег, им приходилось рецензировать статьи, при написании которых, по-видимому, отчасти использовался ИИ.
- И эти статьи содержали в некоторых математических леммах вопиющие ошибки.

Взаимодествие с генеративным ИИ

- В этом плане представляется перспективным сочетание систем генеративного ИИ с системами интерактивного формального доказательства.
- Это могло бы быть интересным и в том случае, когда системы ITP играют роль фильтров, отсеивающих неудачные “доказательства”, сгенерированные ИИ.
- И, возможно, на этапе обучения генеративного ИИ.

THANKS FOR YOUR ATTENTION!

References

1. Очень обширный список ссылок на литературу можно найти в статье которая была использована при подготовке данного выступления:
2. Research report Proof assistants for teaching: a survey Frederic Tran Minh, Laure Gonnord, and Julien Narboux,LCIS, Grenoble-INP. 2024. hal-04705580, https://hal.science/hal-04705580v1
3. Сайты, где можно самому интерактивно работать с Coq: https://coq.vercel.app/, https://jscoq.github.io/wa/
4. Сайт TouIST: https://www.irit.fr/TouIST/